111 — Install repair guard program instructions

↓

112 — Identify repair guard client code block

↓

113 — Identify random number R

↓

114 — Create copy of repair guard's client code block

↓

115 — Disguise client code block copy

↓

116 — Store disguised copy of client code block

↓

117 — Damage original client code block

FIG. 1A

121 — Execution of application software program begins

122 — Damaged client code block loaded into computer memory

123 — Repair guard program instructions are executed

124 — Damaged client code block overwritten by undisguised client code block

125 — Application software program execution proceeds using repaired client code block

FIG. 1B

(a) Without a guard...

```
...
jump important_target
...
...
mem[%r] := mem[%r] + 1
...
```

(b) With a guard...

```
...
client:
    jump important_target
...
...
    mem[%r] += mem[client]
    mem[%r] += mem[%r] - k
...
```
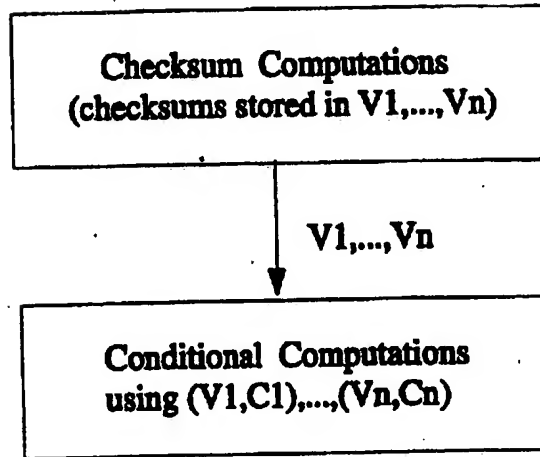
FIG. 2A

```
┌─────────────────────────────────────┐
│     Checksum  Computations          │
│  (checksums stored in V1,...,Vn)    │
└─────────────────────────────────────┘
                    │
                    │  V1,...,Vn
                    ▼
┌─────────────────────────────────────┐
│     Conditional  Computations       │
│  using (V1,C1),...,(Vn,Cn)          │
└─────────────────────────────────────┘
```

FIG. 2 $\mathcal{B}$

```
// template 1

        movl        $RANDOM_1, CHECKSUM_1 // RANDOM_1 = any integer
        movl        $START_1, TEMP_1
LABEL_1:
        Cmpl        $END_1, TEMP_1
        jg          LABEL_2
        addl        (TEMP_1), CHECKSUM_1
        addl        $RANDOM_2, TEMP_1 // RANDOM_2 in [3,5]
        jmp         LABEL_1
LABEL_2:


//template2

        movl        $START_1+END_1+RANDOM_1, TEMP_1 // RANDOM_1 = any
                    integer
        xorl        CHECKSUM_1, CHECKSUM_1
        movl        TEMP_1, CHECKSUM_2
LABEL_1:
        addl        -END_1-RANDOM_1(TEMP_1), CHECKSUM_1
        xorl        -END_1-RANDOM_2+3(TEMP_1), CHECKSUM_2
        subl        $-RANDOM_2, TEMP_1// RANDOM_2 in [3,5]
        cmpl        $END_1+END_1+RANDOM_1, TEMP_1
        jle         LABEL_1
```

FIG. 3

$$(x + 3)(y + 7) \tag{1}$$
$$= \quad (x + 3 + 0)(y + 7 + 0) \tag{2}$$
$$= \quad (x + 3 + u - u_0)(y + 7 + w_0 - w) \quad \text{iff } u = u_0 \text{ and } w = w_0 \tag{3}$$
$$= \quad (x + u + k_1)(y - w + k_2) \quad \text{where } k_1 = 3 - u0 \text{ and } k_2 = 7 + w_0 \tag{4}$$
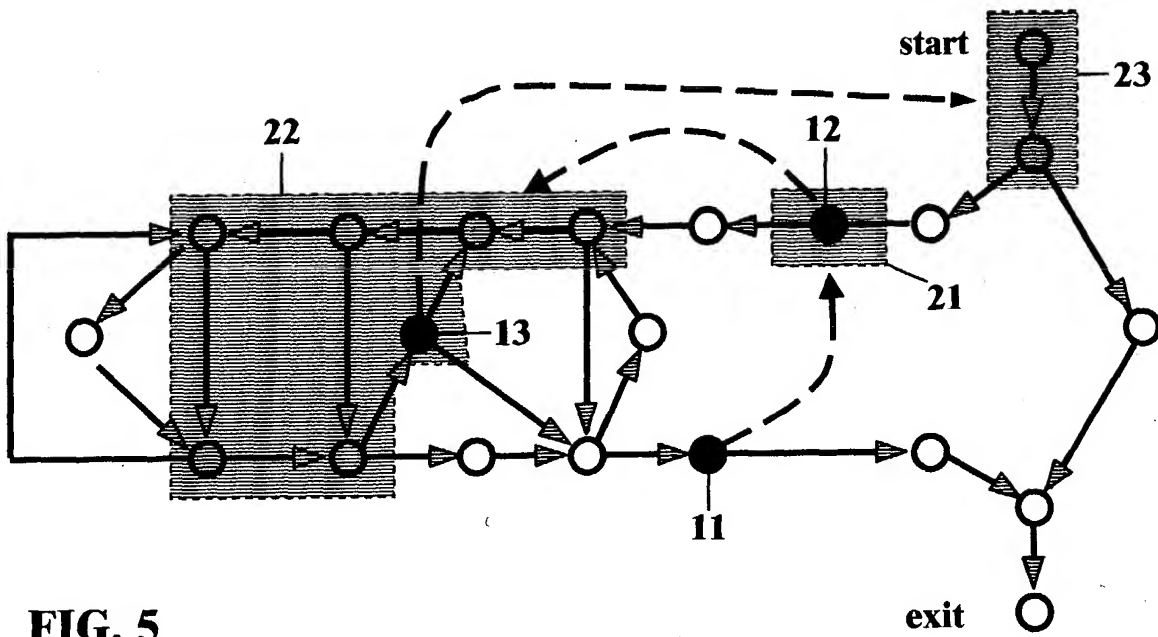
FIG. 4

FIG. 5

```
 1    main:
 2            leal      -4(%esp), %esp        // %esp := %esp – 4
 3            movl      %ebp, (%esp)          // mem[%esp] := %ebp
 4            movl      %esp, %ebp
 5            subl      $8, %esp              // %esp := %esp – 8
 6            leal      -8(%ebp), %eax
 7            leal      -4(%esp), %esp
 8            movl      %eax, (%esp)
 9            leal      -4(%esp), %esp
10            movl      $str1, (%esp)
11            leal      -4(%esp), %esp
12            movl      $next1, (%esp)        // mem[%esp] := $next1
13            jmp       scanf                 //jump to location scanf
14
15    next1:
16            addl      $8, %esp              // %esp :=%esp + 8
17            leal      -4(%esp), %esp
18            movl      -8(%ebp), %eax        // %eax := mem[%ebp-8]
19            movl      %eax, (%esp)
20            leal      -4(%esp), %esp
21            movl      $next2, (%esp)
22            jmp       pr_fact
23
24    next2
25            addl      $4, %esp
26            movl      %ebp, %esp
27            movl      (%esp), %ebp
28            leal      4(%esp), %esp
29            leal      4(%esp), %esp
30            jmp       *-4(%esp)
31
32    pr_fact:
33            leal      -4(%esp), %esp
34            movl      %ebp, (%esp)
35            movl      %esp, %ebp
36            subl      $4, %esp
37            movl      $1, -4(%ebp)
38            jmp       .L94
39
40    .L94
41            cmpl      $1, 8(%ebp)           // mem[%ebp+8] – 1 == ??
42            g         .L96                  // if ?? > 0, jmp .L96
43            jmp       .L98
44
45    .L96
46            movl      -4(%ebp), %eax
47            imul      8(%ebp), %eax         // %eax := %eax * mem[%ebp+8]
48            movl      %eax, -4(%ebp)
49            subl      $1, 8(%ebp)
50            jmp       .L94
51
52    .L98
53            leal      -4(%esp), %esp
54            movl      -4(%ebp), %eax
55            movl      %eax, (%esp)
56            leal      -4(%esp),
```

FIG. 6A

```
57              movl      $str2, (%esp)
58              leal      -4(%esp), %esp
59              movl      $next3, (%esp)
60              jmp       printf
61
62    next3:
63              addl      $8, %esp
64              movl      %ebp, %esp
65              movl      (%esp), %ebp
66              leal      4(%esp), %esp
67              leal      4(%esp), %esp
68              jmp       *-4(%esp)              // jump to addr in mem[%esp-4]
69
```

FIG. 6B

```
1    main:
2            leal      -4(%esp), %esp
3            movl      %ebp, (%esp)
4            movl      %esp, %ebp
5            subl      $8, %esp
6            leal      -8(%ebp), %eax
7            leal      -4(%esp), %esp
8            movl      %eax, (%esp)
9            leal      -4(%esp), %esp
10           movl      $str1, (%esp)
11           leal      -4(%esp), %esp
12           movl      $next1, (%esp)
13           jmp       scanf
14
15   next1:                                   // start of client
16           addl      $8, %esp
17           leal      -4(%esp), %esp
18           movl      -8(%ebp), %eax
19           movl      %eax, (%esp)
20           leal      -4(%esp), %esp
21           movl      $next2, (%esp)
22           jmp       pr_fact
23   end1:                                    // end of client
24
25   next2:
26           addl      $4, %esp
27           movl      %ebp, %esp
28           movl      (%esp), %ebp
29           leal      4(%esxp), %esp
30           leal      4(%esp), %esp
31           jmp       *-4(%esp)
32
33   pr_fact:
34           leal      -4(%esp), %esp
35           movl      %ebp, (%esp)
36           movl      %esp, %ebp
37           subl      $4, %esp
38
39           // guard installation site
40
41           movl      $100, g1
42           movl      $next1, %eax
43   guard1_1:
44           cmpl      $end1, %eax
45           jg        guard1_2
46           movl      g1, %ecx
47           addl      (%eax), %ecx
48           movl      %ecx, g1
49           addl      $3, %eax
50           jmp       guard1_1
51   guard1_2:
52
53           // end of checksumming: (g1, G!)
54
55           movl      $-G1+1, %eax           // G1 is the checksum constant
56           addl      g1, %eax
```

FIG. 7A

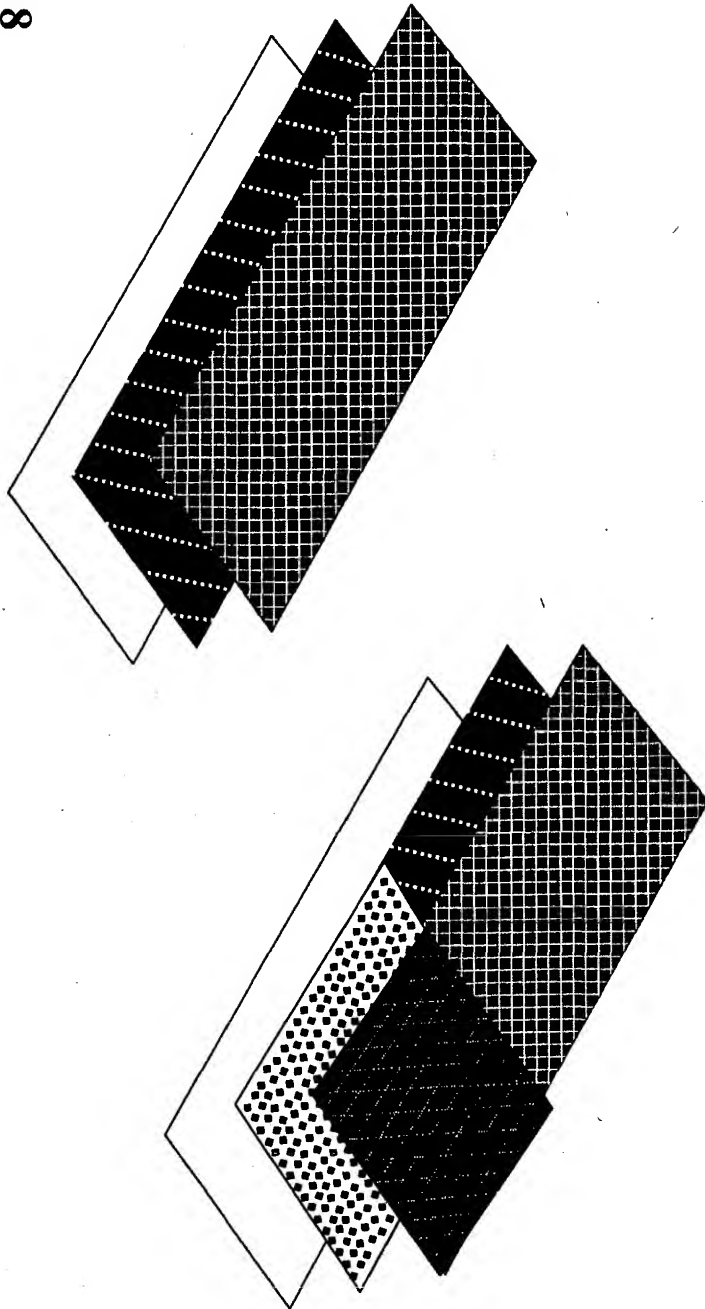```
57   movl   %eax, -4(%ebp)
58   jmp   .L94
59
60   .L94:
61   leal   8+G1(%ebp), %eax
62   subl   g1, %eax
63   movl   (%eax), %eax
64   cmpl $1, %eax
65   jg   .L96
66   jmp   .L98
67
68   .L96:
69   movl   -4(%ebp), %eax
70   imul   8(%ebp), %eax
71   movl   %eax, -4(%ebp)
72   subl $1, 8(%ebp)
73   jmp   .L94
74
75   .L98:
76   leal   -4(%esp), %esp
77   movl   -4(%ebp), %eax
78   movl   %eax, (%esp)
79   leal   -4(%esp), %esp
80   movl $str2, (%esp)
81   leal   -4(%esp), %esp
82   movl $next3, (%esp)
83   jmp   printf
84
85   next3:
86   addl $8, %esp
87   movl   %ebp, %esp
88   movl   (%esp), %ebp
89   leal   4(%esp), %esp
90   leal   4(%esp), %esp
91   jmp   *-4(%esp)
92
```
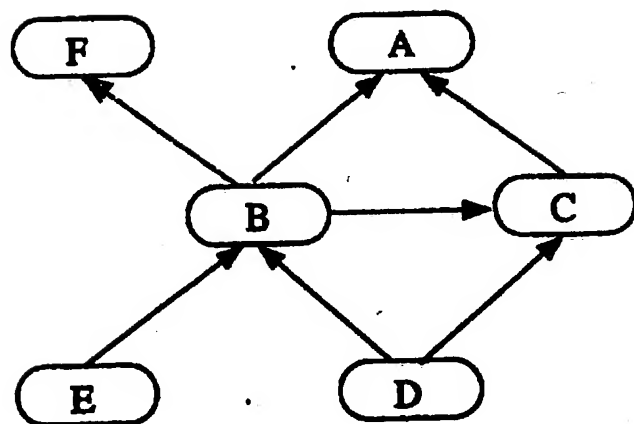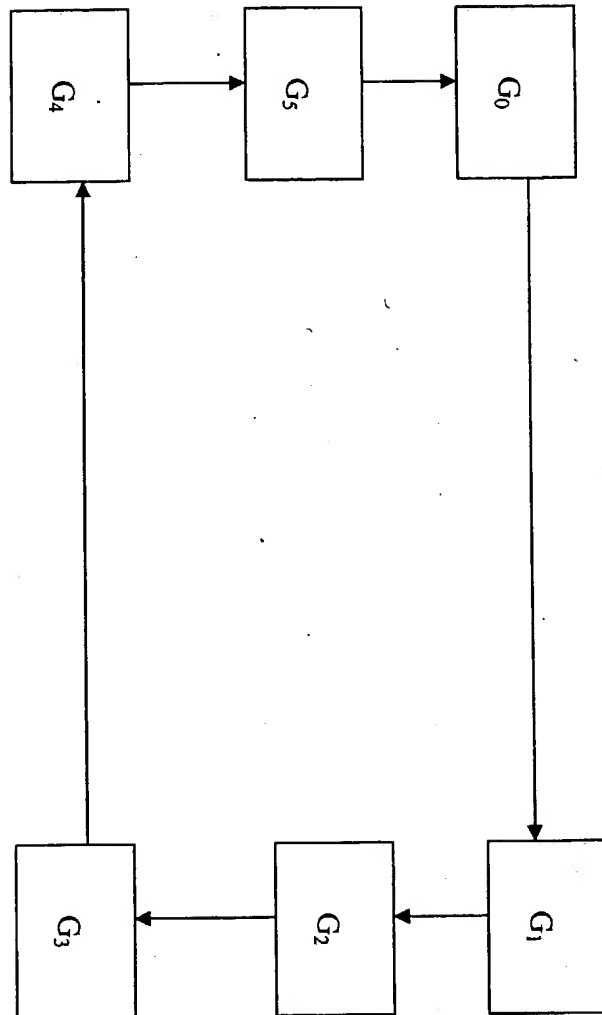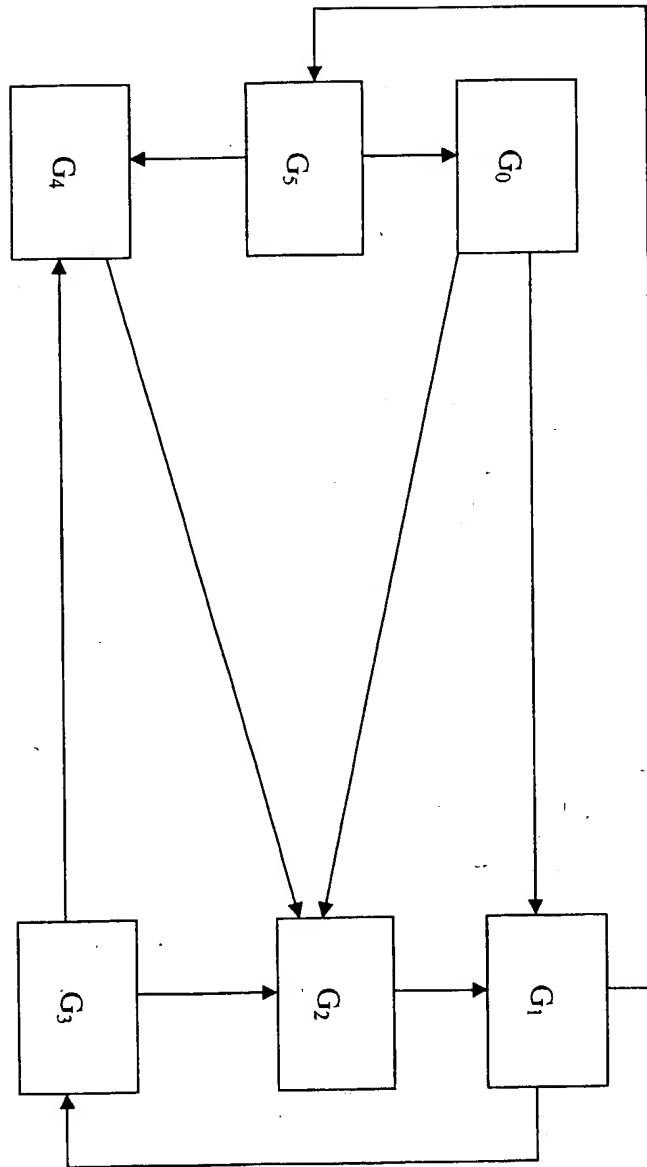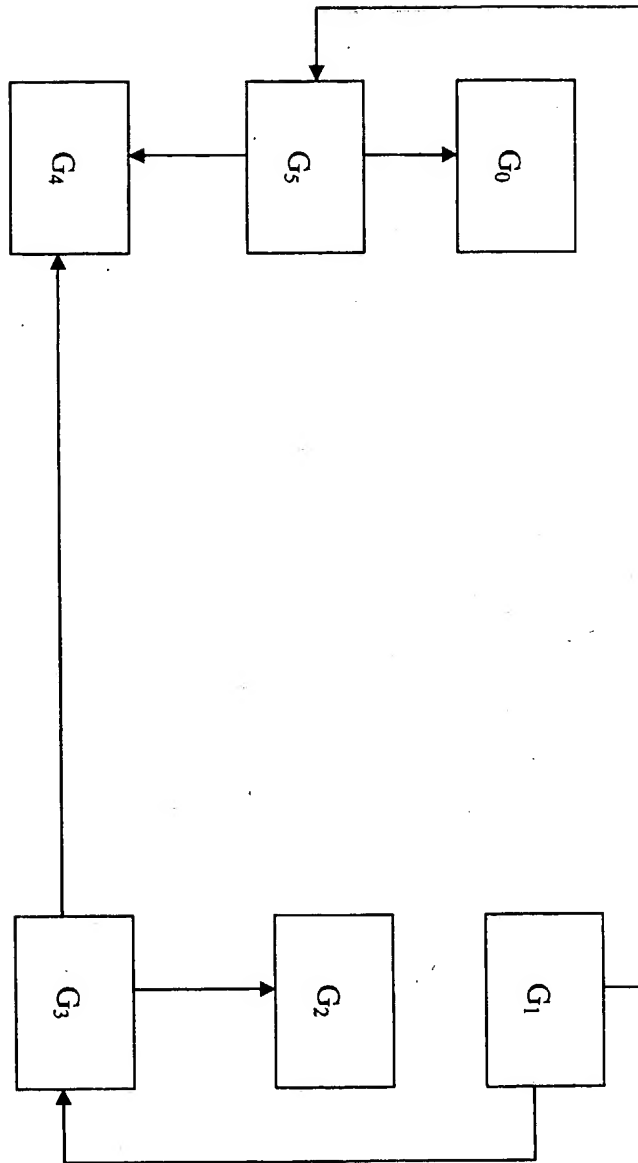
FIG. 7B

FIG. 8

**FIG. 9** A

**FIG. 9B**

**FIG. 9C**

**FIG. 9D**

**FIG. 9E**

G₄  G₅  G₀

G₃  G₂  G₁

**FIG. 9F**

G1

G2

C1

G5

checksums

repairs

checksums

C2

checksums

repairs

G3

G4

**(a) Memory layout of the guarded program**

G3

repairs

checksums                    checksums

C1

checksums

G1

checksums

G5

checksums

repairs

checksums

C2                    G4                    G2

checksums

**(b) The corresponding guard graph**

FIG. 9G

**FIG. 9H**

**FIG. 9I**

CFG-merging

CFG-cloning

Data-aliasing

No

Enough?

Yes

**FIG. 10**

FIG. 11

g1 = <12,20>
g2 = <$f_body,$g_body>

**f:**
```
pushl %ebp
movl %esp,%ebp
subl $ 12,%esp
movl $f_body,%eax
jmp *eax
```
f_body

**+**

**g:**
```
pushl %ebp
movl %esp,%ebp
subl $ 20,%esp
movl $g_body,%eax
jmp *eax
```
g_body

**=**

**f:**
**g:**
```
pushl %ebp
movl %esp,%ebp
subl g1,%esp
movl g2,%eax
jmp *eax
```
f_body    g_body

**FIG. 12**

```
 1   main:
 2   movl $38, g2   // g2=38   <38>
 3   movl $main_2+30, %eax
 4   subl   g2, %eax
 5   movl   %eax, g3   // g3=main_2+30-g2   <main_2-8>
 6   jmp   main_1
 7   // g2=<32>, g3=<main_2-8>
 8   main_1:
 9   pr_fact:
10   leal   -4(%esp), %esp
11   subl $30, g2   // g2=g2-30   <8,4>
12   movl   %ebp, (%esp)
13   movl   g2, %ebp
14   addl   %ebp, g3   // g3=g3+g2   <main_2,pr_fact_1>
15   movl   %esp, %ebp
16   subl   g2, %esp
17   movl   g3, %eax
18   jmp   *%eax
19   // g1=<,?> g2=<8,4>, g3=<main_2,pr_fact_1>
20   main_2:
21   addl $26, g2   // g2=g2+26   <34>
22   leal   -8(%ebp), %eax
23   leal   -4(%esp), %esp
24   movl   %eax, (%esp)
25   leal   -4(%esp), %esp
26   movl $str1, (%esp)
27   leal   -4(%esp), %esp
28   movl $next1, (%esp)
29   jmp   scanf
30   // g2=<34>
31   next1:   // client start
32   addl $8, %esp
33   leal   -4(%esp), %esp
34   movl $pr_fact_1-38, %eax
35   addl   g2, %eax
36   movl   %eax, g3   // g3=g2+pr_fact_1-38 <pr_fact_1-4>
37   movl   -8(%ebp), %eax
38   movl   %eax, (%esp)
39   leal   -4(%esp), %esp
40   movl $next2, (%esp)
41   jmp   pr_fact
42   // g2=<34>, g3=<pr_fact_1-4>
43   end1:   // client end
44   next2:
45   addl $4, %esp
46   movl   %ebp, %esp
47   movl   (%esp), %ebp
48   leal   4(%esp), %esp
49   leal   4(%esp), %esp
50   j,mp   *-4(%esp)
51   pr_fact_1:   // guard installation site
52   movl   , $100, g1
53   movl $next1, %eax
54   jmp   guard1_1
55   // g1=<?>
56   guard1_1:
```
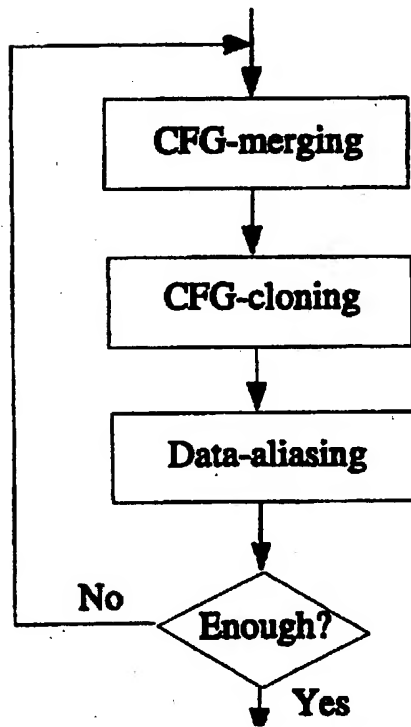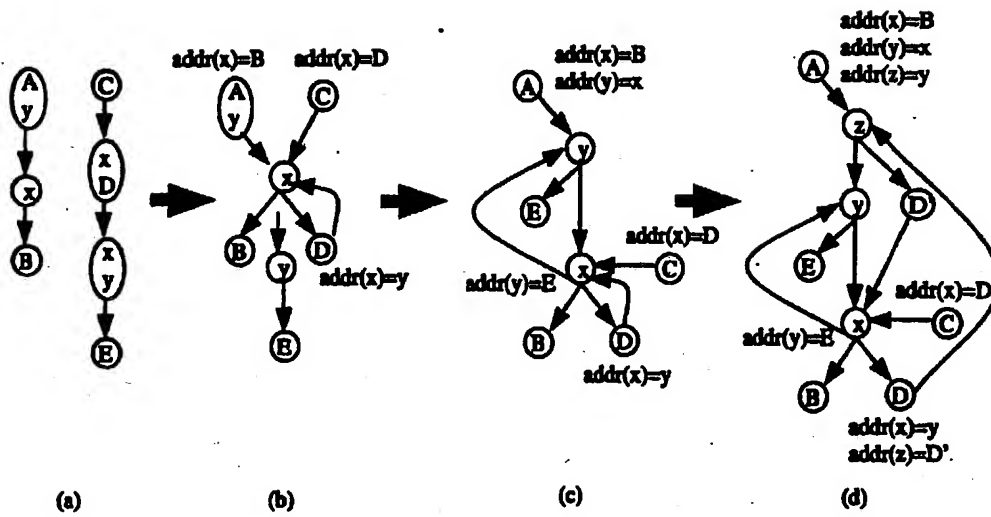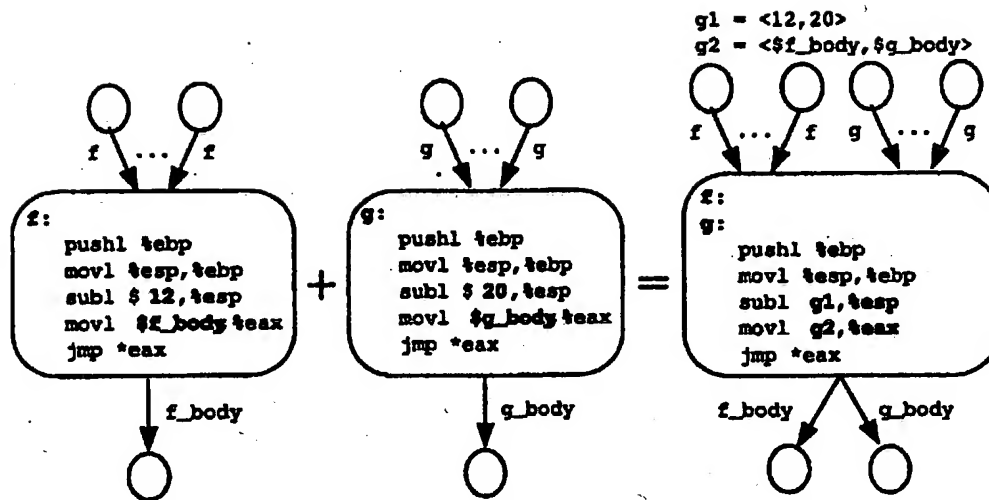
FIG. 13A

```
57              cmpl        $end1, %eax
58              jg          guard1_2
59              jmp         guard1_3
60              // g1=<?>
61      guard1_3:
62              movl        g1, %ecx
63              addl        (%eax), %ecx
64              movl        %ecx, g1
65              addl        $3, %eax
66              jmp         guard1_1
67              // g1=<?>
68      guard1_2:
69              // end of checksumming: (g1, G1)              // g1=<G1>
70              movl        $-G1+1, %eax
71              addl        g1, %eax
72              movl        %eax, -4(%ebp)
73              jmp         .L94
74              // g1=<G1>
75      .L94:
76              leal        8+G1(%ebp), %eax
77              subl        g1, %eax
78              movl        (%eax), %eax
79              cmpl        $1, %eax
80              jg          .L96
81              jmp         .L98
82              // g1=<G1>
83      .L96:
84              movl        -4(%ebp), %eax
85              imul        8(%ebp), %eax
86              movl        %eax, -4(%ebp)
87              subl        $1, 8(%ebp)
88              jmp         .L94
89      .L98:
90              leal        -4(%esp), %esp
91              movl        -4(%ebp), %eax
92              movl        %eax, (%esp)
93              leal        -4(%esp), %esp
94              movl        $str2, (%esp)
95              leal        -4(%esp), %esp
96              movl        $next3, (%esp)
97              jmp         printf
98      next3:
99              addl        $8, %esp
100             movl        %ebp, %esp
101             movl        (%esp), %ebp
102             leal        4(%esp), %esp
103             leal        4(%esp), %esp
104             jmp         *-4(%esp)
105
```

FIG. 13B

```
1    main:
2            movl    $38, g2                    // g2=38              <38>
3            movl    $scanf-1000, g5            // g5=scanf-1000      <scanf-1000>
4            movl    $main_2+30, %eax
5            subl    g1, %eax
6            movl    %eax, g3                   // g3=main_2+30-g2    <main_2-8>
7            jmp     main_1
8            // g2=<38>, g3=<main_2-8>, g5=<scanf-1000>
9    main_1:
10  · pr_fact:
11           leal    -4(%esp), %esp
12           subl    $30, g1                    // g2=g2-30           <8,4>
13           movl    %ebp, (%esp)
14           movl    g2, %ebp
15           addl    %ebp, g3                   // g3=g3+g2           <main_2,pr_fact_1>
16           movl    %esp, %ebp
17           subl    g2, %esp
18           movl    $1000, %eax
19           addl    g5, %eax
20           movl    %eax, g4                   // g4=g5+1000         <scanf>
21           movl    g3, %eax
22           jmp     *%eax
23           // g1=<,?> g2=<8,4>, g3=<main_2,pr_fact_1., g4=<scanf,>
24   main_2:
25           addl    $26, g2                    // g2=g2+26           <34>
26           movl    $next1-794320, %eax
27           addl    g2, %eax
28           movl    %eax, g1                   // g1=g2+next1-794320  <next1-794286>
29           leal    -8(%ebp), %eax
30           leal    -4(%esp), %esp
31           movl    %eax, (%esp)
32           leal    -4(%esp), %esp
33           movl    $str1, (%esp)
34           jmp     main_2_1
35           // g1=<next1-794286>, g2=<34>, g4=<scanf>
36   next1:   // client start
37           addl    $8, %esp
38           leal    -4(%esp), %esp
39           movl    $pr_fact_1-38, %eax
40           addl    g2, %eax
41           movl    %eax, g3                   // g3=g2+pr_fact_1-38  <pr_fact_1-4>
42           addl    $next2-794286-next1, g1    // g1=g1+next2-794286-next1 <next2-794286>
43           movl    $pr_fact, g4               // g4=pr_fact          <pr_fact>
44           movl    -8(%ebp), %eax
45           movl    %eax, (%esp)
46           jmp     next1_1
47           // g1=<next2-794286>, g2=<34>, g3=<pr_fact_1-4>, g4=<pr_fact>
48   next1_1:
49   main_2_1:
50   .L98_1:
51           leal    -4(%esp), %esp
52           addl    $794286, g1                // g1=g1+794286        <next1>
53           movl    g1, %eax
54           movl    %eax, (%esp)
55           movl    g4, %eax
56           jmp     *%eax
```

FIG. 14A

```
57                    // g1=<next2,next1,next3>, g2=<34,34,>, g3=<pr_fact_1-4,,>,
58                    // g4=<pr_fact,scanf,printf>, g5=<?>
59     end1:          // client end
60     next2:
61            addl    $4, %esp
62            movl    %ebp, %esp
63            movl    (%esp), %ebp
64            leal    4(%esp), %esp
65            leal    4(%esp), %esp
66            jmp     *-4(%esp)
67     pr_fact_1:     // guard installation site
68            movl    $100, g1              //                              <?>
69            movl    $next1, %eax
70            jmp     guard1_1
71                    // g1=<?>
72     guard1_1:
73            cmpl    $end1, %eax
74            jg      guard1_2
75            jmp     guard1_3
76                    // g1=<?>
77     guard1_3:
78            movl    g1, %ecx
79            addl    (%eax), %ecx
80            movl    %ecx, g1
81            addl    $3, %eax
82            jmp     guard1_1
83                    // g1=<?>
84     guard1_2:
85                    // end of checksumming: (g1,G1)
86            movl    $printf-G1, %eax
87            addl    g1, %eax
88            movl    %eax, g4             // g4=g1+printf-G1
89            movl    $-G1+1, %eax
90            addl    g1, %eax
91            movl    %eax, -4(%ebp)
92            jmp     .L94
93                    // g1=<g1>, g4=<printf>
94     .L94:
95            leal    8+G1(%ebp), %eax
96            subl    g1, %eax
97            movl    (%eax), %eax
98            cmpl    $1, %eax
99            jg      .L96
100           jmp     .L98
101                   // g1=<g1>, g4=<printf>
102    .L96
103           movl    -4(%ebp), %eax
104           imul    8(%ebp), %eax
105           movl    %eax, -4(%ebp)
106           subl    $1, 8(%ebp)
107           jmp     .L94
108                   // g4=<printf>
109    .L98:
110           addl    $-G1+next3-794286, g1    // g1=g1-G1+next3-794286     <next3-794286>
111           leal    -4(%esp), %esp
112           movl    -4(%ebp), %eax
```

FIG. 14B

```
113          movl       %eax, (%esp)
114          leal       -4(%esp), %esp
115          movl       $str2, (%esp)
116          jmp        .L98_1
117          // g1=<next3-794286>, g4=<printf>
118   next3:
119          addl       $8, %esp
120          movl       %ebp, %esp
121          movl       (%esp), %ebp
122          leal       4(%esp), %esp
123          leal       4(%esp), %esp
124          jmp        *-4(%esp)
125
```
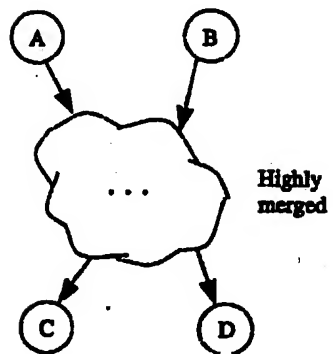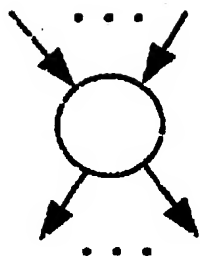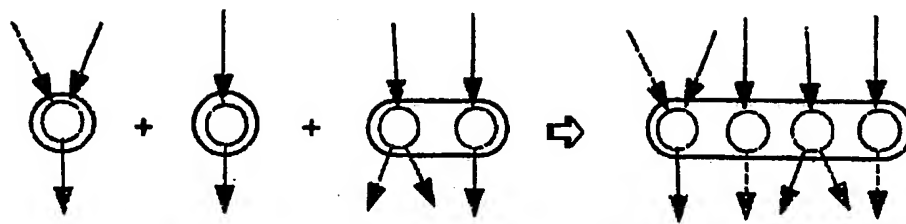
FIG. 14C
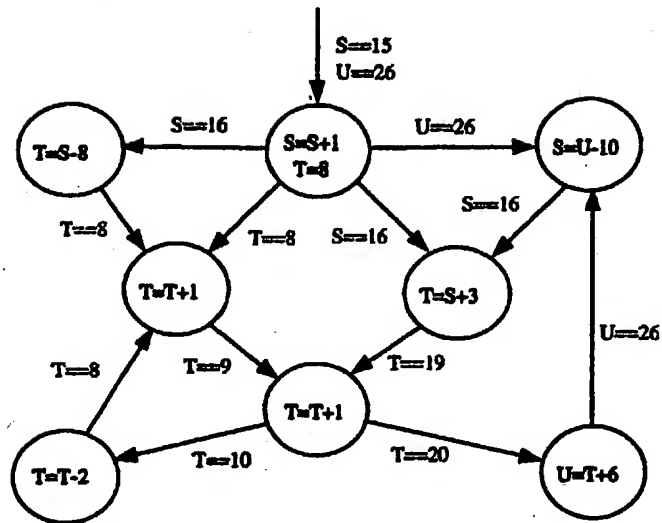
FIG. 15

FIG. 16

FIG. 17

FIG. 18

Procedure precompute (variable v, link-nodes $\{x\_1,...,x\_n\}$,
$\qquad$ constants $\{c\_1,...,c\_n\}$)

$\quad$ Set live(v,x_i) = TRUE and value$(v,x\_i) = c\_i$ for all $i = 1,...,n$;

$\quad$ While (there is a link-node x and a variable v such that
$\qquad$ value(v,x) is defined but done(v,x)==FALSE)

$\qquad$ Let X = $\{x\_1,...,x\_n\}$ be the entire set of link-nodes in the
$\qquad\qquad$ same basic block, say B, that contains x;
$\qquad$ Let t be the point within B for inserting evolve( );

$\qquad$ If (no evolve( ) has previously been chosen for B)
$\qquad\qquad$ Choose a mathematical function evolve(U), where U is a (possibly
$\qquad\qquad$ empty) set of (new) global variables, with the following properties:
$\qquad\qquad$ (1) $\quad$ No u in U is reserved (i.e. no u in U that, for some z in pred-
$\qquad\qquad\qquad$ links(X), live(u,z)==TRUE but value(u,z) is undefined);
$\qquad\qquad$ (2) $\quad$ For any x_i in X where value(v,x_i) is defined, the common
$\qquad\qquad\qquad$ evolve(U) is able to fulfill value(v,x_i) by setting value(u,z)
$\qquad\qquad\qquad$ appropriately, for any u in U and z in pred-links(x_i) with
$\qquad\qquad\qquad$ unseen(u,z,t)==TRUE;
$\qquad\qquad$ (3) $\quad$ For all x_i in X where value(v,x_i) is undefined, the same
$\qquad\qquad\qquad$ evolve(U) is possible to fulfill any desired value for any future
$\qquad\qquad\qquad$ definition of value(u,x_i);

$\qquad$ End if

$\qquad$ If (evolve( ) is newly chosen and is not the trivial identity function)
$\qquad\qquad$ Insert code at t of B for computing v=evolve(U);
$\qquad$ End if

$\qquad$ For (all u in U and z in pred-links(x) where unseen(u,z,t)==TRUE)
$\qquad\qquad$ Set live(u,z) = TRUE and value(u,z) equal to some values such that these
$\qquad\qquad$ new values, together with other values of U already set along the paths to t
$\qquad\qquad$ through z, satisfy value(v,x)==evolve(U);
$\qquad$ End for

$\qquad$ For (each x_i in X where value(v,x_i) is underfined, and each u in U and z in
$\qquad$ pred-links(x_i) where unseen(u,z,t)==TRUE)
$\qquad\qquad$ Set live(u,z) = TRUE;
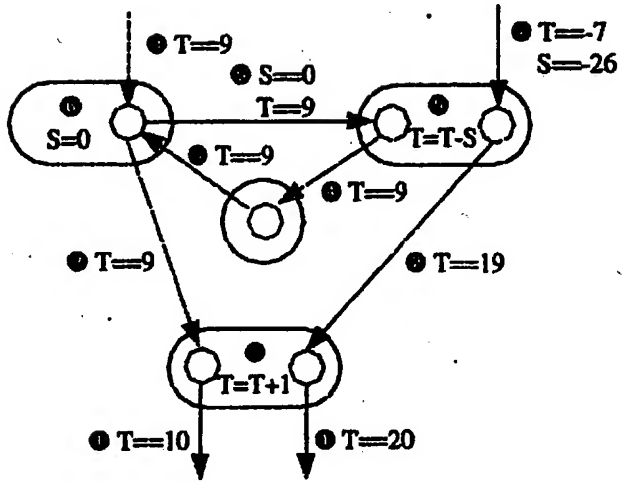$\qquad$ End for

$\qquad$ Set done(v,x) = TRUE;
$\qquad$ End while
End procedure
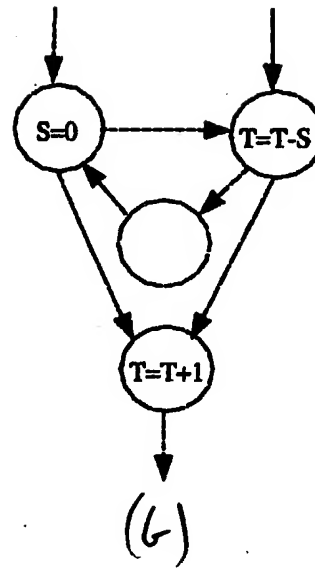
FIG. 19A

Procedure unseen(variable v, link-node z, program point t)

    If (value(v,z) is undefined, and at the basic block that contains t, v is not defined in the code before t)

        return TRUE;

    Else

        return FALSE;

    End procedure

FIG. 19B

(a)

(b)

FIG. 20

addr(x)=B
addr(y)=x
addr(z)=y

addr(y)=E

addr(x)=D

addr(x)=y
addr(z)=D'

FIG. 21

```
// (a) before change
        . . .
    jmp            .L24
```

```
// (b) after change
        . . .
    cmpl           %eax, (%esp)        // args are randomly selected
    jl             .L013               // clone
    jmp            .L24
```
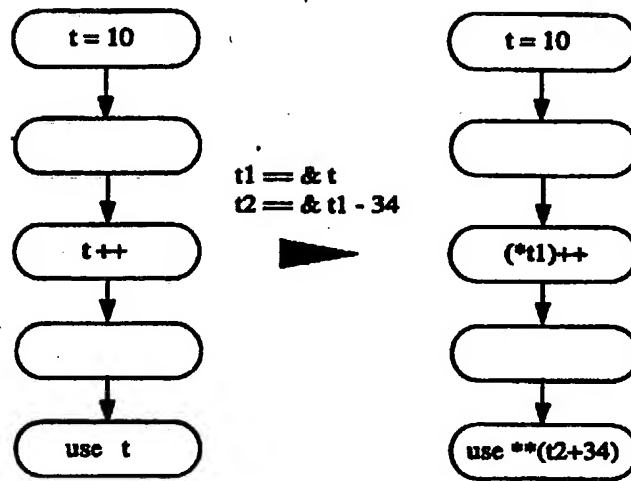
FIG. 22

t1 = & t
t2 = & t1 - 34

FIG. 23

tag	sh f	E(len + sh f)	E(msg + sh f)

FIG. 24

FIG. 25

```
┌─────────────────────────────────────────────────┐
│           Preprocess the input code             │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│     Install self-protecting mechanisms to the code │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│     Embed watermarks and produce a program file │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│  Assemble the file and link it with other resources (if any) │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│           Patch the file with data values       │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│        Remove symbol tables from the file        │
└─────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────┐
│      Attach additional information to the file   │
└─────────────────────────────────────────────────┘
```

FIG. 26

High-level instructions	Simpler instructions
call *operand* next: . . .	pushl $next jmp *operand* next: . . .
ret	leal 4(%esp), %esp jmp * −4(%esp)
enter	pushl %ebp movl %esp, %ebp
leave	movl %ebp, %esp popl %ebp
pushl *operand*	leal −4(%esp), %esp movl *operand*, (%esp)
pop1 *operand*	movl (%esp), *operand* leal 4(%esp), %esp

FIG. 27

Software Program Code	Encrypted customization parameters	Digital signature

FIG. 28

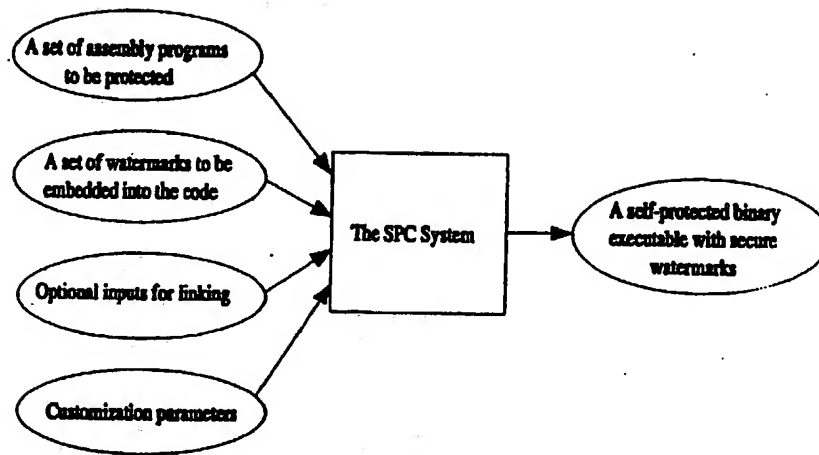| Wrong password	After "bypassing" password
password: **abc** Invalid password!	password: **abc** **n ? 100000** next prime = 100001
Right password	After further "corrections"
password: **opensesame** **n ? 100000** next prime = 100003	password: **abc** **n ? 100000** Segmentation Fault (core dumped)

FIG. 29

FIG. 30